Patent Application
Docket No. 92717-294

CERTIFICATE OF MAILING BY EXPRESS MAIL

"EXPRESS MAIL" Mailing Label No.: EL524961505US
Date of Deposit: October 30, 2000

Type or Print Name: CAROL MARSTALLER

Signature

09702454 - 103000

INSIGHT ARCHITECTURE VISUALIZATION TOOL

BACKGROUND OF THE INVENTION

Technical Field of the Invention

The present invention is related to architecture visualization and, more particularly, to architecture visualization during conceptual, development or deployment phases of a system.

Description of Related Art

Architecture plays a critical part in the success of any development. Distributed systems are no exception, where architecture plays a key role throughout the lifetime of the system, from concept, to prototype, to partial system during development increments, and even in the complete deployed

system. Before implementation of a system begins, it enables a model to be developed, verified and stress tested with use cases scenarios, enabling the design of the system to be fine-tuned up front without incurring the cost of changing code. During system implementation, the architecture provides a reference from which subsystems and services may be developed with confidence that they will integrate and collaborate with the rest of the system when complete. Even during deployment, the architecture plays an important role in guiding system changes and enhancements during its lifetime. Implementations in the form of components and middlewares typically change in any system with a lifespan of longer than a few years, while the architecture of the system lives on.

However, it is often the case that users are not fluent in software architectures and concepts. Furthermore, this effect is compounded by the fact that there is still no one pervasive standard architectural language throughout the software industry. This dilemma effectively reduces the ability of the user to participate in system design, development and maintenance, and limits the extent to which they are able to leverage the benefits of a solid and

thorough architecture. This dilemma has far-reaching primary consequences including the following: (1) less participation in early stages of design, leading to late changes and "scope creep"; (2) an incomplete understanding of system complexity causing inaccurate planning; and (3) limited ability to foresee the consequence of system changes leading to reduced flexibility, extensibility, reliability, scalability and increased system downtime.

Secondary effects of this dilemma in a best case scenario involve increased costs and delays, and in a worst case scenario involve project failure. This dilemma presents a challenge for system developers to give users more insight into architectures during all phases of a system, from prototype to deployment.

The crucial role that architecture plays in any complex development cannot be disputed. However, a lack of understanding of architectural concepts, compounded by the lack of a pervasive standard architectural language often limits the realization of the benefits of a solid architecture. As such, visualization is a powerful paradigm that may be used to convey architectural ideas and concepts effectively.

In view of the above, it should be understood that the visualization of architectures facilitates a better understanding of architectures and, consequently, an improved ability to leverage the associated investment and benefits.

Accordingly, it is therefore an object of the present invention to provide a communication tool to help a user of a distributed system to visualize and understand architecture and behavior required to realize business use cases before system implementation begins.

It is also an object of the present invention to provide a diagnostic tool to monitor and verify system behavior as it is being built.

It is a further object of the present invention to provide a management tool for use during deployment to monitor a system.

SUMMARY OF THE INVENTION

The present invention is directed to a system and method for visualizing architectures both statically and dynamically. One or more graphical views of the tiers and

components of a multi-tier architecture are presented and, accordingly, the system events are monitored.

BRIEF DESCRIPTION OF THE DRAWINGS

A more complete understanding of the method and apparatus of the present invention may be acquired by reference to the following Detailed Description when taken in conjunction with the accompanying Drawings wherein:

FIGURE 1 illustrates an exemplary view of an Application Architecture;

FIGURE 2 illustrates a Simple View Example;

FIGURE 3 illustrates a Configuration Root Node which may contain three children that include the Model 16, View 12 and Controller 14 elements;

FIGURE 4 illustrates a Configuration Model Node which may contain the ArchitectureModel element that in turn contains the TierModelGroup, PathModelGroup and EventModelGroup elements;

FIGURE 5 illustrates a Configuration TierModelGroup Node which may contain one or more TierModel elements;

FIGURE 10 illustrates a Configuration TierView Node which may contain one or more TierView elements;

FIGURE 10A describes the valid attributes for the DisplayName element;

FIGURE 10B describes the valid attributes for the TierView element;

FIGURE 10C describes the valid attributes for the ComponentView element;

FIGURE 11 illustrates a Configuration EventView Node which may contain zero or more EventView elements;

FIGURE 11A describes the valid attributes for the EventView element;

FIGURE 11B describes the valid attributes for the PathView element;

FIGURE 11C describes the valid attributes for the ComponentHighlightView element;

FIGURE 12 illustrates a Configuration Controller Node which determines the "feel" or behavior of an application;

FIGURE 13 illustrates an exemplary view of an Application Architecture for Viewing in Simulation Mode;

FIGURE 14 illustrates a Normal End to End Scenario for Viewing an Architecture in Simulation Mode;

FIGURE 5A describes the valid attributes for the TierModel element;

FIGURE 5B describes the valid attributes for the ComponentModel element;

FIGURE 6 illustrates a Configuration PathModelGroup Node which may contain zero or more PathModel elements;

FIGURE 6A describes the valid attributes for the PathModel element;

FIGURE 7 illustrates a Configuration EventModelGroup Node which may contain zero or more EventModel elements;

FIGURE 7A describes the valid attributes for the Event element;

FIGURE 7B describes the valid attributes for the Property element;

FIGURE 8 illustrates a Configuration View Node which may contain the information required for a presentation;

FIGURE 9 illustrates a Configuration ArchitectureView Node which may contain one or more ArchitectureView elements;

FIGURE 9A describes the attributes that may be added to the ArchitectureView element;

FIGURE 9B describes the optional view properties for Basic View implementation;

FIGURE 15 illustrates an exemplary view of an Application Architecture for Viewing in Auto-Cycle Mode;

FIGURE 16 illustrates a Normal End to End Scenario for Viewing an Architecture in Auto-Cycle Mode;

FIGURE 17 illustrates an exemplary view of an Application Architecture for Viewing a Deployed Implementation of an Architecture; and

FIGURE 18 illustrates a Normal End to End Scenario for Viewing a Deployed Implementation of an Architecture.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

The numerous innovative teachings of the present application will be described with particular reference to the presently preferred exemplary embodiments. However, it should be understood that this class of embodiments provides only a few examples of the many advantageous uses of the innovative teachings herein. In general, statements made in the specification of the present application do not necessarily delimit any of the various claimed inventions. Moreover, some statements may apply to some inventive features but not to others.

Essentially, in accordance with the present invention, an Application Architecture has a standard Model-View-Controller (MVC) architecture, which is composed of a plurality of components. A detailed description of each component of this architecture and examples of their roles, responsibilities and collaborations are provided below. Note that the implementations of the components are configurable via the XML 20 configuration and read at startup time. Note also that all of the components inside the box are running in the same process.

Specifically, FIGURE 1 illustrates an exemplary view of an Application Architecture 10. As shown, the exemplary architecture includes a View 12 component, a Controller 14 component, a Model 16 component, an Event Service Delivery Agent 18, an Event Service Interface 15 and an XML 20 configuration.

The Model 16 component manages information regarding the modeled architecture including tiers, components, communication paths and events. This component receives events via the generic middleware and protocol independent event service interface, and may selectively map these events to the view where they are visualized. However, the Model

and Event Service Interfaces 15 that the Model 16 component uses to listen for events are not pluggable. These interfaces are the core of the application.

The View 12 component manages the presentation that determines the "look" of the application. Note that different looks are required for different uses of the present invention. For example, in a prototype or demonstration mode, the view implementation that is plugged into the present invention through an XML 20 configuration change should purposely slow down event visualization to give users time to understand the system behavior. On the other hand, in a mode where the present invention is monitoring a real deployed system, the view cannot afford to slow down the visualization of events, of which hundreds may be arriving per second. In this case, the view implementation plugged into the present invention may more appropriately visualize system behavior based on event frequency, for example, using different path colors or thicknesses to represent communication path activity. The View 12 component receives visualization events from the Model 16 component and displays these as specified in the XML 20 configuration. It also

receives GUI events from the user and forwards these to the Controller 14 component where necessary.

The Controller 14 component manages behavior of the application, effectively determining its "feel." This component receives events from the Model 16 and View 12 components, and responds by collaborating with the Model 16 or View 12 component to handle these events. The particular implementation of the Controller 14 component is specified in the XML 20 configuration for the application and may therefore be swapped in order to change the "feel" of the application.

The Event Service Delivery Agent 18 manages how the event service is delivered to the application. This component is based on the TRC Service Delivery Framework, as illustrated in FIGURE 2. This The Event Service Delivery Agent 18 is implemented as a single class with minimal responsibilities, since it is the component that is switched to adapt the present invention to any system that uses any middleware or protocol. At initialization time, this component is responsible for locating the local or remote implementation of the event service to which it will delegate requests. At runtime, this component is responsible for

receiving events and forwarding them in the form of callbacks on the Model 16 component. If the event service implementation is remote, this delegation may involve the bidirectional translation of types from the generic Event Service Interface 15 (shown in FIGURE 1) to the middleware or protocol specific types of the remote event service implementation. The particular implementation of this component is defined in the XML 20 configuration and may therefore be swapped in order to adapt the present invention to different uses or systems. For example, in a standard middleware known as the Common Object Request Broker Architecture (CORBA) system, the delivery agent may collaborate with the CORBA Event Service using Internet Inter-ORB Protocol (IIOP). The Object Request Broker (ORB) provides all the communication infrastructure needed to identify and locate objects, handle connection management and deliver data and request communication. However, in an MQ Series based system, the delivery agent may collaborate with the system via message queues.

With further reference to FIGURE 2, there is illustrated an architecture that facilitates mobile distributed computing with XML. In this Simple View Example, the illustrated

architecture is used to deliver product browsing services that allow clients to specify a category of product and browse all products available in the given category. The concepts presented here, however, may be applied to deliver other more complex services in the same way. A detailed description of the actors and components in this architecture, in particular their roles, responsibilities, and how they collaborate to deliver the same service to a variety of clients in a variety of different formats, is provided below.

Three actors are shown as "tier one clients" 210 in the architecture. The first actor, the Mobile User, is using a PalmVII PDA 211 to access services wirelessly and a Wireless Application Protocol (WAP) Web Phone 212 to easily access services and information instantly over the Internet. Similarly, the Web User is using a standard PC running a web browser 213, such as Netscape or Internet Explorer, to access services directly over the Internet. Likewise, the External Server 214 represents a server that accesses services directly over the Internet and represents a dependent server in an extranet or collaborative e-commerce network. Note that many other types of tier one clients may interact in

this architecture. All tier one clients may communicate with the Web Server using either Hypertext Transport Protocol (HTTP) or secure HTTP (HTTPS) as specified by the application.

The BellSouth Tower 221 component is a cellular base station in the Wireless Infrastructure 220. It receives requests for services in the form of wireless signals sent from a Palm VII PDA 211, forwards these requests to the Palm.net data center for processing, and finally relays the results back to the Palm VII 211. Note that the requests and replies sent between the Palm VII PDA 211 and the base station 221 are secure and compressed. The Palm.Net Data Center Proxy component (not shown) is a bank of proxy servers that receive requests for Internet-based services from clients 210 via base stations. The proxy first decrypts and decompresses each request and then forwards it to the target Web Servers 231 for processing. Finally, the proxy compresses and encrypts the replies and then forwards them back to the clients 220, again via the base stations. Note that the communications between the proxy and Web Servers 231 may use either HTTP or HTTPS, as specified by the application. The format of the information in this case is

a proprietary "Web Clipping" format defined by Palm Computing that is specialized for the Palm VII 211 mobile device.

The Web Server 231 component is responsible for receiving requests from any type of client 210 over the Internet via either HTTP or HTTPS, and either handling or dispatching these requests for processing. In the case of the example architecture, the Web Server 231 dispatches incoming requests to servlets 240 that act as gateways 230 to back-end distributed systems. The JRun Servlet Engine component (not shown) is responsible for hosting application servlets 240 and providing them with a standards-compliant servlet interface. A key advantage of using JRun instead of running the servlets 240 inside the Web Server process itself, is that JRun provides a standard servlet API and decouples the servlets 240 from the specific implementation details of the Web Server 231. This buys architectural flexibility in facilitating swapping of the Web Server 231 with no impact on the overall system. The servlet engine also provides various management and debugging features and shifts the load of servlet processing away from the Web Server 231.

The Product Data Servlet 251 component is a gateway servlet responsible for receiving requests from the Web Server 231 for product information and forwarding these requests to the CORBA Product Service 261 using Internet Inter ORB Protocol (IIOP) for processing. After processing, it receives the results from the CORBA Product Service 261, reformats them in the form of XML, and forwards them back to the Web Server 231 for delivery to the client 210. Note that the CORBA Naming Service 262 is used by this servlet 251 at startup to locate the CORBA Product Service 261.

The Product Presentation Servlet 241 component is another gateway servlet responsible for receiving requests from the Web Server 231 for product information from various types of clients 210. It identifies the type of client 210, either explicitly from parameters forwarded by the client with the request, or implicitly from header information associated with the HTTP or HTTPS request. It forwards the request to the Product Data Servlet 251 using a servlet chaining approach and receives the results of the request in the form of XML. The servlet 251 then retrieves an XSL template 242 from the XSL Transformations repository (not shown) for the identified client 210 and uses it to

automatically reformat the XML results into a format appropriate for the given client 210. Finally, the servlet 251 forwards the reformatted product results back to the client 210 via the Web Server 231.

The XSL Transformations repository component stores the XSL templates 242 used to automatically transform the product information results from the base XML format into a format appropriate for a particular client 210.

The XML Data repository component (not shown) stores static content and provides access to it in XML format. This is in contrast to dynamic content in XML format generated "on the fly", for example, by the Product Data Servlet 251. The XML Data repository enables the architecture illustrated in Figure 2 to expose both static and dynamic content for use either in its base XML or a derived form. For example, in one scenario the Web Server 231 may publish static XML data from the XML Data repository directly to an External Server 214. Alternatively, the XML Data repository may provide static XML data to a presentation servlet (for example, the Product Presentation Servlet 241) for transformation according to an XSL template 242 into a format appropriate for a particular type of client 210.

The CORBA Naming Service 262 component provides directory-based publication and lookup services. When CORBA servers (for example, the CORBA Product Service 261) start up, they may publish themselves to the CORBA Naming Service to facilitate subsequent lookup by CORBA clients (for example, the Product Data Servlet 251). This component provides location transparency in the CORBA distributed system. The CORBA Product Service 261 component is a CORBA business service 260 responsible for publishing itself to the CORBA Naming Service 262 at startup, and subsequently processing requests from CORBA clients for product information.

The present invention is highly configurable. A configuration is specified using XML 20. This configuration is read during initialization at application startup and includes both high level and detailed configuration information. High level configuration information includes, for example, the particular implementations of the components to use to implement the View 12, Controller 14 and Event Service Delivery Agent 18 components, while more detailed information may be split into the four categories outlined as follows: (1) abstract information in the form of tiers,

components, communication paths and events may be configured in the XML 20 to model an architecture of arbitrary complexity; (2) presentation information in the form of how many display views are required to present the architecture, and how to respond visually when events are received; (3) controller information that may specify details that determine how the particular controller implementation behaves; and (4) integration information that may be used by the particular implementation of the Event Service Delivery Agent 18. This information may be required, for example, in order for the agent to collaborate with the system being monitored.

The goal in modeling an architecture is to create a configuration in XML 20 format that describes a model of the architecture to visualize, as well as how to present it. The configuration is represented in XML 20 that is both well-formed and valid according to a Document Type Definition (DTD). It is the elements of the DTD that define the scope of configurability of the present invention. In this discussion, graphical views of the DTD elements will be used to concisely and clearly present their meaning. In this graphical presentation, each XML element or tag is

represented as a blue box. Elements to the left are parent of elements to the right to which they are connected using red lines. Elements in the same vertical tier that have the same parent are siblings. These relationships translate in a straightforward manner into the well established tag nesting paradigm established in XML 20.

Referring now to FIGURE 3, there is illustrated a Configuration Root Node 300. The root element of any configuration in the present invention is the node that contains three children that include the Model, View and Controller elements. As their name implies, there is a close relationship between these elements and the application architectures as shown, for example, in Figure 1.

Referring now to FIGURE 4, there is illustrated a Configuration Model Node 400. The Model element contains the ArchitectureModel element that in turn contains the TierModelGroup, PathModelGroup and EventModelGroup elements. The TierModelGroup element contains abstract (non-presentation) information required to model the tiers of the architecture including, for example, the names of the tiers and the components they contain. All tiers of the architecture are configured here, however, note that the

present invention may be configured to view only a subset of these at a time as shown subsequently in the discussion of the view configuration.

A communication path is defined here as a collaboration between any two components in the architecture. The PathModelGroup element is configured with the abstract information regarding all communication paths in the architecture, even though only a subset of these paths may be shown at a time in any given view.

The EventModelGroup element is used to specify the details of all events to which the present invention will listen and respond visually.

Referring now to FIGURE 5, there is illustrated a Configuration TierModelGroup Node 500. The TierModelGroup element contains one or more TierModel elements, each of which is configured with a ComponentModelGroup element that in turn contains one or more ComponentModel elements. Note that all components in the tier are configured here. However, the present invention may be configured to view only a subset of these components at a time. FIGURE 5A and FIGURE 5B describe the valid attributes for the TierModel 500A and ComponentModel 500B elements respectively.

Referring now to FIGURE 6, there is illustrated a Configuration PathModelGroup Node 600. The PathModelGroup contains zero or more PathModel elements. Each communication path is configured using a PathModel element and has a startpoint and an endpoint. Each of these points is specified at an abstract level by a tier and component where the names of the tier and component in each case are configured in the value of the TierName and ComponentName elements respectively. These tiers and components refer to tiers and components specified previously in the configuration under the TierModel elements. FIGURE 6A describes the valid attributes of the PathModel 600A element.

Referring now to FIGURE 7, there is illustrated a Configuration EventModelGroup Node 700. The EventModelGroup element contains zero or more EventModel elements, each of which defines an event that the present invention recognizes. Each event that the present invention listens to through the generic Event Service interface shown in FIGURE 2 is specified by an event channel name in the value of the EventChannelName element, and zero or more name/value property pairs specified in the Property element(s) contained by the PropertyGroup element. The present invention will

only consider a given event to have occurred when an event is received from the named event channel and the received event has the specified name/value property pairs. Note that a received event may optionally have more name/value property pairs, but must have at least the configured name/value properties for the present invention to trigger an internal event based on the received event. In this way, the present invention may be configured to filter events and only respond to events of interest to the viewer. FIGURE 7A and FIGURE 7B describes the valid attributes for the Event 700A and Property 700B elements respectively.

Referring now to FIGURE 8, there is illustrated a Configuration View Node 800. The View element contains the information required for a presentation. It must contain a JavaClass element that specifies the implementation of the view to use. For example, to use the basic view bundled with the present invention, one may set the value of this element to com.trcinc.tools.insight.view.basic.ViewImplementation.

This basic view lays out components and tiers in a simple grid layout, the cell size of which is determined automatically by the largest component used in the view(s).

To change the view implementation, one may implement an alternative view, for example in com.trcinc.tools.insight.view.myview.MyViewImplementa-tion. Any valid view implementation class must implement the com.trcinc.tools.insight.view.ViewListener interface and must be public.

The background texture of the present invention view may optionally be specified with the BackgroundTileIconUrl element, the value of which should contain a valid URL that refers to the image to tile in the background of the view behind the architecture tiers. For example, a gray textured background is used in the view shown in FIGURE 2.

An icon may optionally be specified in the IconUrl element for a logo to go on the view at the top right. The value of this element is again a valid URL that refers to the image for the icon to use. For example, the TRC logo is used in the view shown in FIGURE 2.

The View element also contains an ArchitectureViewGroup element that in turn may contain one or more ArchitectureView elements. Each ArchitectureView element corresponds to a tab of the final view. This enables the present invention to show any architecture of arbitrary complexity by splitting it up into multiple views. The tiers and components shown

in any given view are generally chosen as the components involved in one or more related use cases. For example, the view show in FIGURE 2 contains a single ArchitectureView element represented in the single tabbed pane with the title "Overview". FIGURE 9A shows the attributes that may be added to the ArchitectureView 900A element. FIGURE 9B shows the Optional View Properties representation which can be used for a basic view implementation 900B.

Zero or more name/value property pairs may also be configured for the View element in the child Property element(s) contained by the PropertyGroup element. The actual name/value pairs used for these elements are arbitrary and may be used to convey specific configuration information required by a particular view implementation. Where more complex configuration information must be conveyed to the View element than may be represented in the form of simple name/value property pairs, these properties may be used to refer to another more sophisticated configuration source, for example, by specifying an URL to another configuration document.

Referring now to FIGURE 9, there is illustrated a Configuration ArchitectureView Node 900. Each tabbed pane

of the present invention view has a tab name specified by the value of the child DisplayTabName element, a title specified by the value of the child DisplayName element, one or more TierView elements contained by the TierViewGroup element, and zero or more EventView elements contained by the EventViewGroup element. Each TierView element specifies a tier of the architecture that must be presented in that view and corresponds to a valid tier of the architecture specified previously using a TierModel element. Each EventView element specifies and event that should generate some visual response in the given view, for example, a path highlight. Each of these events correspond to valid events configured previously using EventModel elements. Furthermore, each of these EventView elements contains all of the information regarding what visual response to display in the given view when the associated event is received. FIGURE 10A describes the valid attributes for the DisplayName element.

Referring now to FIGURE 10, there is illustrated a Configuration TierView Node 1000. For each TierView element, the corresponding displayed tier contains a title specified as the value of the DisplayName child element. Zero or more components may also be displayed using the ComponentView

element(s) contained by the ComponentViewGroup element. Each displayed component may optionally have a title specified by the value of the DisplayName child element. A displayed component may also optionally have icons for its inactive and active states specified by the value of IconUrl and HighlightIconUrl child elements respectively. If an element is not assigned a title or any icons, then it will be invisible on the display. This is a valid use case, for example, where one wants to insert space between components in a tier. FIGURE 10A, FIGURE 10B and FIGURE 10C discuss the valid attributes for the DisplayName 1000A, TierView 1000B and ComponentView 1000C elements respectively.

Referring now to FIGURE 11, there is illustrated a Configuration EventView Node 1100. The EventViewGroup node may contain zero or more EventView elements that may each be used to specify visual behavior that occurs as a result of an associated event being received. A text status message may optionally be specified in the value of the Message child element. This message is displayed in the status bar at the bottom of the present invention display whenever an instance of the associated event is received. The main visual behavior that the present invention shows when an event is

received is communication path and associated component highlights. Zero or more PathView child elements contained by the PathViewGroup element may be used to specify the exact visual behavior that must occur when a new event is received. PathView child elements that are siblings of the same parent node will be shown sequentially. In other words, visualization associated with the first PathView element will complete before visualization associated with the second sibling PathView element commences and so forth. Note that PathView elements may also be nested. In other words, a PathView element may contain another PathView element. In this case, the highlighting associated with the parent PathView element will remain active until the highlighting associated with all of its child elements has completed. This parent/child usage of PathView elements is typically used for visualization in the case of a nested synchronous type of communication. The PathView element refers to valid PathModel elements specified previously. Sometimes it is desirable to have a component remain highlighted independent of a path highlight, for example, in an asynchronous store and forward type of communication. In this case, a ComponentHighlightView element contained by the

ComponentHighlightViewGroup element may be used to specify the start and end of the component highlight. Note that both the start and end points of the highlight must be fixed with respect to the point in time at which the associated event arrives. FIGURE 11A, FIGURE 11B and FIGURE 11C describe the valid attributes of the EventView 1100A, PathView 1100B and ComponentHighlightView 1100C elements respectively.

Referring now to FIGURE 12, there is illustrated a Configuration Controller Node 1200. The controller determines the "feel" or behavior of the application. The particular implementation of the controller to use is configurable and specified by the value of the JavaClass child element. For example, to use the basic controller implementation bundled with the present invention, the value of this element would be set to com.trcinc.tools.insight.controller.basic.Controller-Implementation.

An alternative controller may be written and plugged into the application. Any valid controller should implement the com.trcinc.tools.insight.con-troller.ControllerListener interface and should be public. Furthermore, a new controller should be put in a sibling package to the basic

package, for example, a new controller could be implemented in com.trcinc.tools.insight.controller.myController.MyControllerImplementation.

Zero or more arbitrary name/value property pairs may be conveyed to the controller implementation via the Property child elements contained by the PropertyGroup element. These properties may be used, for example, to convey specific information required by a particular implementation of a controller. Where more complex configuration information is required than may be represented in simple name/value property pairs, these properties may be used to refer to a more sophisticated configuration source. For example, a property may be used to specify an URL for another XML configuration document required by the controller implementation.

Referring now to FIGURE 13, there is illustrated an exemplary view of an Application Architecture for Viewing in Simulation Mode 30. As shown, the exemplary architecture 30 includes a Companion Tool 11, a View 12 component, a Controller 14 component, a Model 16 component, an XML 20 configuration, a Local Event Service Delivery Agent 19 and a Local Event Service 21.

The goal of this use case 30 is to communicate specific architectural collaborations before implementation, for example, during the conceptual and prototype stage before development starts.

Referring now to FIGURE 14, there is illustrated a Normal End to End Scenario for viewing an Architecture in Simulation Mode 1400. There are two key differences in this use case 30:

The companion tool is implemented and runs in the same process as the application. At initialization time, the companion tool reads the same XML 20 configuration as the present invention and, therefore, knows the events that the present invention is listening to. The companion tool then displays a GUI with buttons that enables the user to manually fire events; and

The implementation of the Event Service Delivery Agent 18 used in this case is the Local Event Service Delivery Agent 19. This particular delivery agent delegates requests from either the Model 16 component or the companion tool to a lightweight implementation of the Local Event Service 21 that is running in the same process.

Referring now to FIGURE 15, there is illustrated an exemplary view of an Application Architecture for Viewing in Auto-Cycle Mode 50. As shown, the exemplary architecture 50 includes an Event Generator 13, a View 12 component, a Controller 14 component, a Model 16 component, an XML 20 configuration, a Local Event Service Delivery Agent 19 and a Local Event Service 21.

The goal of this use case 50 is to communicate all modeled architectural collaborations in a continuous, free-running mode, for example, in demonstrations at conferences.

Referring now to FIGURE 16, there is illustrated a Normal End to End Scenario for viewing an Architecture in Auto-Cycle Mode 1600. here are two key differences in this use case 50:

The Event Generator 13 is implemented. Like the companion tool in the previous use case, the Event Generator 13 reads the XML 20 configuration as the present invention at startup. The Event Generator 13, therefore, also understands the events that the present invention is listening to. It computes the intervals and order in which to fire the events and then begins firing these events in a continuous repetitive cycle; and

The implementation of the Event Service Delivery Agent 18 used in this case is the same Local Event Service Delivery Agent 19 that again delegates to the lightweight implementation of the Local Event Service 21 running in the same process.

Referring now to FIGURE 17, there is illustrated an exemplary view of an Application Architecture for Viewing a deployed implementation of an Architecture 70. As shown, the exemplary architecture 70 includes a View 12 component, a Controller 14 component, a Model 16 component, a CORBA Event Service Delivery Agent 23, a CORBA Event Service 22, an XML 20 configuration and an N-Tier Distributed System 24.

The goal of this use case 70 is to visualize real system behavior, for example, during development, demonstrations or general system administration during deployment.

Referring now to FIGURE 18, there is illustrated a Normal End to End Scenario for viewing a Deployed CORBA Implementation of an Architecture 1800. There is one key difference in this use case 70:

The implementation of the Event Service Delivery Agent 18 used is the CORBA Event Service Delivery Agent 23. At startup, this agent locates the remote standard CORBA

Event Service 22. At runtime, this agent receives requests from the Model 16 component, reformats generic types associated with the requests to CORBA types where necessary, and delegates these requests to the remote event service. For example, this occurs when the Model 16 component subscribes the event channels it will monitor. At runtime, the delivery agent also receives callbacks from the CORBA Event Service 22, for example, when new events are published. Where CORBA specific types or exceptions are present in collaborations between the agent and event service, the agent also serves to convert these types to and from generic types that may be returned to the Model 16 component through the generic Event Service Interface 15. Note that the CORBA Event Service Delivery Agent 23 uses only the untyped push support provided by the CORBA Event Service 22.

Note that, although in this case, CORBA is used for the middleware and implementation of the event service, any other middleware and event service implementation could be used through suitable implementation of an associated Event Service Delivery Agent 18. The XML 20 configuration for the present invention could then be updated to specify the use of this new agent.

In this way, the present invention may be adapted to any system that uses any middleware or protocol to communicate.

The current implementation of the present invention uses the Java JDK1.2 and is under the package com.trcing.tools.insight, which includes sub-packages. It depends on the TRC Component Framework that lives under the package com.trcinc.framework, and in particular the TRC Service Delivery Framework that lives under com.trcinc.frameworks.delivery. The XML4J Java XML API from IBM is used to parse XML configuration documents. Only the standard interface of the XML4J API is used. Therefore any other standards compliant XML API could be used in plase of XML4J.

The basic view implementation that is bundled with the present invention is designed specifically for demonstrations and conferences and supports use during prototyping, development and deployment as long as event generation is controlled either directly using the companion tool, or indirectly through some application client. This view shows a purposely delayed response to incoming events in order to give users time to understand the nature of the collaboration that is taking place. For this reason, this view is not

suitable for visualizing behavior of a real deployed system that may generate hundreds or thousands of events per second. For this, another view should be written and may be "plugged" into the application as previously discussed in the first use case. The classes that implement the basic view may be found in the com.trcinc.tools.insight.view.basic Java package. A new view implementation should be introduced as a sibling to the basic package. For example, an xxx view should be added under the package com.trcinc.tools.insight.view.xxx.

The basic controller implementation bundled with the present invention is lightweight, and since the viewer is primarily output rather than input biased, the controller does not have a lot of responsibility. The basic controller implementation lives under the com.trcinc.tools.insight. controller.basic Java package. A new controller implementation should be introduced as a sibling to the basic package. For example, an xxx controller should be added under the package com.trcinc.tools.insight.controller.xxx.

It should be understood that both the present invention and companion tool are highly configurable tools that may be used to visualize any architecture with any view and may be

adapted to any system implementation, regardless the middleware implementation used for that system.

Although the presently preferred embodiments are directed toward systems and methods that facilitate system developers in the modeling of an architecture through XML 20 configuration to increase understanding of system architectures and behaviors through visualization, it should also be understood that the principles of the present invention may be implemented in a variety of different uses or systems, e.g., in a CORBA system.

As will also be recognized by those skilled in the art, the innovative concepts described in the present application can be modified and varied over a wide range of applications. For example, although the embodiments are shown in a distributed environment, the invention is not limited to such an architecture and can be practiced in other application architectures, such as a message queuing application.

Accordingly, the scope of the present invention should not be limited to any of the specific exemplary teachings discussed, but is only limited by the following claims.

A list of tables and screen displays for the monitoring software and technique of the present invention are attached.